Discovering Individual and Collaborative Problem-Solving Modes with Hidden Markov Models

Fernando J. Rodríguez(✉) and Kristy Elizabeth Boyer

Department of Computer Science, North Carolina State University,
Raleigh, NC 27695, USA
{fjrodri3,keboyer}@ncsu.edu

Abstract. Supporting students during learning tasks is the main goal of intelligent tutoring systems, and the most effective systems can adapt to students based on a model of their current state of knowledge or their problem-solving actions. Most tutoring systems focus on individual students, but there is growing interest in supporting student pairs. However, modeling student pairs involves considerations that may differ from individual students. This paper reports on hidden Markov models (HMMs) of student interactions within a visual programming environment. We compare HMMs for individual students to those of student pairs and examine the different approaches the students take. The resulting models suggest that there are some important differences across both conditions. There is potential for using these models to predict problem-solving modes and support adaptive tutoring for collaboration in problem-solving domains.

Keywords: Collaboration · Hidden markov models · Pair programming · Visual programming

1 Introduction

Intelligent tutoring systems (ITSs) support student learning by adapting problem difficulty [1], providing personalized hints [2, 3], or giving feedback on the learner's progress [4]. However, ITSs' ability to support problem solving have traditionally been limited when the problem is a creative or open-ended learning task because such problems have many possible correct solutions [4, 5]. While some lines of research are actively investigating complex domains whose problems do not have formulaic solutions [6], much work is needed in order to seamlessly support users in solving open-ended problems. Nonetheless, this type of learning task support is an essential step in supporting human learning with computers, as solving open-ended problems is a central component of many real-life endeavors.

In addition to the limitation of most tutoring systems to well-defined tasks, the vast majority of adaptive learning environments focus on supporting individual users. However, collaborative problem solving is not only mandated within curricular standards for a variety of disciplines, it arises organically both inside and outside classroom settings [7]. When people solve problems collaboratively, their approaches and strategies differ from individual problem solving [8], and developing an understanding of those differences is crucial for adaptively supporting collaborative problem solving.

C. Conati et al. (Eds.): AIED 2015, LNAI 9112, pp. 408–418, 2015.
DOI: 10.1007/978-3-319-19773-9_41

This paper represents a step toward automatically supporting collaborative pairs in a complex problem-solving domain. We examine paired problem solving as it arises when solving computer science problems, within structured collaboration referred to as pair programming [9]. We present an exploratory study of the differences in problem-solving approaches between individual students and student pairs. Based upon detailed interaction logs of the problem-solving collaboration, we built hidden Markov models of the interaction sequences for both individual students and collaborative student pairs. While not predictive of learning gains, the models highlight that many of the problem-solving modes are shared across both groups; however, each exhibits a different mode that may have emerged as a result of working individually or with a partner. These findings point the way toward building adaptive support for paired problem solving.

2 Related Work

Many ITSs build a representation of the student's progress, which allows the system to adapt to that specific student's needs and provide appropriate scaffolding [10]. Aspects of a student's problem-solving process that have been utilized to create student models within ITSs range from low-level actions, such as textual edits [3], to combinations of several features of an action, such as the action type, involved components, and final outcome [11]. The research community has taken an interaction-based perspective, attempting to model learners based on their interactions within an ITS's interface. In particular, models of problem-solving strategies have been utilized to inform the design of adaptive systems [1] by building hidden Markov models of students' problem-solving strategies within an ITS for middle school algebra [12] and concept mapping within a teachable agent for middle school science [13]. The present work builds upon this prior research by focusing on computer science as the learning domain and on comparing individual students to students working in pairs.

The vast majority of ITSs have historically focused on individual students. Although this is still the norm, recent years have shown an increased interest in adapting ITSs to support student pairs and groups in general. In computer science, pair programming is an effective paradigm for supporting learning [9, 14–16]. Even at the elementary school level, student pairs collaborating through an ITS achieved similar learning gains to individual students and were able to do so having completed fewer problems within the tutor [8]. System logs and audio recordings have been used to detect when students are collaborating, as well as the intensity of the collaboration [17]. A framework for detecting common patterns of collaboration based on students' speech and interactions within a touch-based tabletop system has also been proposed [18]. The work presented in this paper contributes to this field of research by presenting models that can be used to compare individual students and student pairs completing a problem-solving task.

3 Study

In this study, students implemented a simple game using a visual programming language, and their collaborative or individual problem-solving actions were recorded.

Students were recruited from the second course on computer programming at North Carolina State University. These students had prior experience with programming in Java, but were pre-screened to ensure that none had interacted with visual programming languages before. A total of 30 students participated in the study: 14 worked individually and 16 were assigned to the paired condition, for a total of 8 student pairs. Students were paired based upon their schedules and then through random selection for mutual availability. Participants' ages varied between 18 and 28 (μ=21), with one participant of age 40. Two of the 30 participants were female, commensurate with the population of computer science undergraduate students at the university; one female student was randomly paired with a male student and the other was in the individual students condition. In exchange for participating in the study, students received credit for a homework assignment in their programming course. The programming task was to implement the win conditions for a game of "Rock-Paper-Scissors." Fig. 1 shows a partially implemented student program and its output.

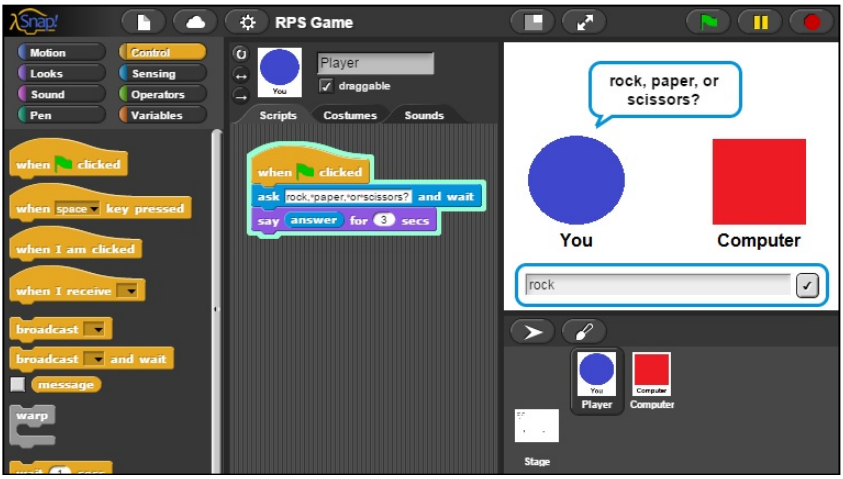

Fig. 1. Visual programming environment and partial rock-paper-scissors program

Students were first given a brief tutorial of the *Snap!* programming language in which program structures are represented by *blocks*, which can be *grabbed*, *dropped*, and *snapped* together to create programs [19]. The students were allowed to ask questions of the research coordinator during and after the initial tutorial, but not during the programming task. Students were provided with two paper artifacts: a scaffolding document for building the program, and a Snap! block reference sheet that included the blocks required to complete the task and where to locate them within the interface. Students working in pairs followed the pair programming paradigm, sharing one computer and performing one of two roles at any given moment during the task: the

driver used the computer and created the program, and the *navigator* read the scaffolding document and provided instructions to the driver. They switched roles three times (at moments indicated within the scaffolding document), giving each student the opportunity to take on both roles twice during the task. Individual students completed the task in an average of 37 minutes ($n_{individuals}$=14; min=23 minutes; max=60 minutes; σ=10.5 minutes), while student pairs tended to complete the task slightly faster, on average 34 minutes (n_{pairs}=8; min=23 minutes; max=42 minutes; σ=6.3 minutes). This difference was not statistically significant (Mann-Whitney *U* test: p=0.73).

Students were given an individual multiple-choice pre-test before the Snap! tutorial and an identical post-test following the task. There were no significant differences in pre-test scores between the individual and collaborative students ($\mu_{individuals}$=0.78; μ_{pairs}=0.81). Overall, students achieved a statistically significant increase in test score over the course of the study (μ_{pre}=0.79; μ_{post}=0.86; one-tailed *t*-test p=0.02). Disaggregating by condition, there was a significant increase in test score for the individual students (μ_{pre}=0.78; μ_{post}=0.89; one-tailed *t*-test p=0.01), but this was not the case for the student pairs (μ_{pre}=0.81; μ_{post}=0.83; one-tailed *t*-test p=0.29).

4 Modeling Interaction Sequences

Having collected the full sequential interaction traces, we proceeded to model the event sequences. The underlying notion is that problem-solving sequences comprise visible actions that reflect problem-solving modes. We use hidden Markov models (HMMs) to model the problem-solving sequences [20]. These represent stochastic processes through states and transitions. At any given moment, the HMM is in one of its hidden states. Each state emits an observation after each event, and these observations can be directly measured and used to predict which state the model is currently in (see Fig. 2). One model was built for each condition of the study: one for individual students and another for student pairs. We interpret the hidden states to represent the "modes" that the students employ during their programming task. (Note that since HMMs model unobservable constructs, the theoretical constructs that they represent are subject to interpretation.) The observations correspond to the event sequences stored in the database of problem-solving logs from the study. Each observation in this model has three dimensions:

- *Event type*: These represent semantically meaningful events within the interface.
  - CREATE: a block was added onto the scripting area
  - DELETE: a block was removed from the scripting area
  - SNAP: two blocks in the scripting area were connected
  - UNSNAP: two blocks in the scripting area were separated
  - MOVE: a block was dragged and dropped within the scripting area
  - PARAM: a block's parameter was set or changed
  - CATEGORY: the current block category was switched
  - RUN: the current program was run

- *Edit distance from final solution*: The textual representation of the session's current program was compared to the final session program (note that each student and student pair completed the task), and edit distance (number of character changes that need to be made to a string to match another string) was calculated. The observation consisted of determining if the student was getting closer or farther from the final solution or remained the same edit distance away from the solution after the event.
- *Elapsed time since previous event*: These were classified as either being under 30 seconds (QUICK) or over 30 seconds (DELAY). This threshold was empirically determined by measuring the elapsed time when the students were either reading the instructions or consulting with their partner.

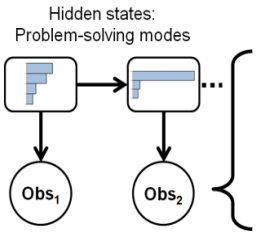

Fig. 2. Sequential HMM diagram

Table 1. Observation symbols

Event	Dist.	Time	Obs. symbols
CREATE	CLOSE	≤ 30s	CREATE_CLOSE_QUICK
DELETE	FAR	(QUICK)	CREATE_CLOSE_DELAY
MOVE	SAME	> 30s	CREATE_FAR_QUICK
SNAP		(DELAY)	…
UNSNAP			DELETE_CLOSE_QUICK
PARAM			…
CATEGORY			CATEGORY_QUICK
RUN			CATEGORY_DELAY
			RUN_QUICK
			RUN_DELAY

With eight event types, two elapsed time tags, and three edit distance tags, there were 48 possible observations from the combinations of these dimensions (see Table 1 for examples). Removing the events that did not occur in the dataset, there were a total of 36 observation symbols. Overall, the individual students carried out an average of 199 task actions to complete the problem-solving task ($n_{individuals}$=14; min=104 events; max=323 events; σ=63.1 events); the student pairs carried out an average of 181 task actions (n_{pairs}=8; min=130 events; max=258 events; σ=41.6 events). The difference between means was not statistically significant (Mann-Whitney U test; p=0.36), and the higher standard deviation for individuals could be attributed to both the differences in programming skills and the lack of a partner to help direct the activity. The events were classified into a set of defined observation symbols based on the elapsed time since the previous event and the edit distance of the current solution from the final session solution, as described in Table 1.

To determine the number of states for the HMMs, leave-one-session-out cross-validation was used. Models were trained using all but one of the problem-solving sessions, and the log-likelihood was calculated for the probability that the remaining session could be generated from the trained model; this was repeated for each session in a given condition (14 times for the individual students, 8 for the student pairs). The same process was repeated for each possible number of states from 3 to 20 (roughly

half the number of the observation symbols), and the average log-likelihood for each number of states was stored. The Bayesian Information Criterion (BIC) was calculated for each number of states and averaged between the two conditions; the lowest average BIC corresponded to the 4-state models. Thus, 4-state HMMs were built for each condition by retraining the models with all sessions for each condition.

5 Results

The HMMs for both conditions are presented in Fig. 3. State transition arrows are thicker for higher probabilities and dotted for lower probabilities. Only the observation probabilities greater than 0.05 are shown in the figure. We named the states based on qualitative interpretation of the observation frequencies. The descriptions for the individual student HMM state interpretations are as follows:

- Block search: frequent category switches
- Program testing and refining: running the program and making changes that take the program closer to its final solution
- Program creation: creating new blocks and snapping them to the existing code, moving closer to the final solution
- Program tweaking: snapping blocks with varying effects to the edit distance, moving blocks and editing parameters without affecting the edit distance to the final solution

The descriptions for the student pair HMM states are as follows:

- Block search: frequent category switches with high frequency of snapping blocks and moving closer to the final solution
- Program testing and refining: running the program and making changes that take the program closer to its final solution
- Program planning: high frequency of category switching with blocks being created and moved within the scripting area to get closer to the final solution
- Program building: moving blocks within the scripting area, snapping them together, and taking them apart (unsnapping) with varying effects to proximity to the final solution

6 Discussion

Through the use of the Viterbi algorithm, in combination with the HMMs and observation sequences, we determined the most likely sequence of hidden states that the individuals and pairs went through during the task. Table 2 shows summary statistics on the percentage of time that individuals spent on each state. In general, most of their time was spent in program tweaking, followed by block search as the second most frequent. As mentioned earlier, category switching was the most common event, which would justify the high frequency of the block search state. The program tweaking state does

not appear to be productive at first, but students may have engaged in off-task behaviors due to being stuck, leading them closer to the solution [21].

Table 2 also shows the summary statistics for the state frequencies of student pairs. Pairs tended to spend more time in the program building state, followed closely by the block search state; these states have high frequency of getting closer to the final solution. The pairs also demonstrated a program planning state, which may be where the collaborative aspect of the task is most prominent.

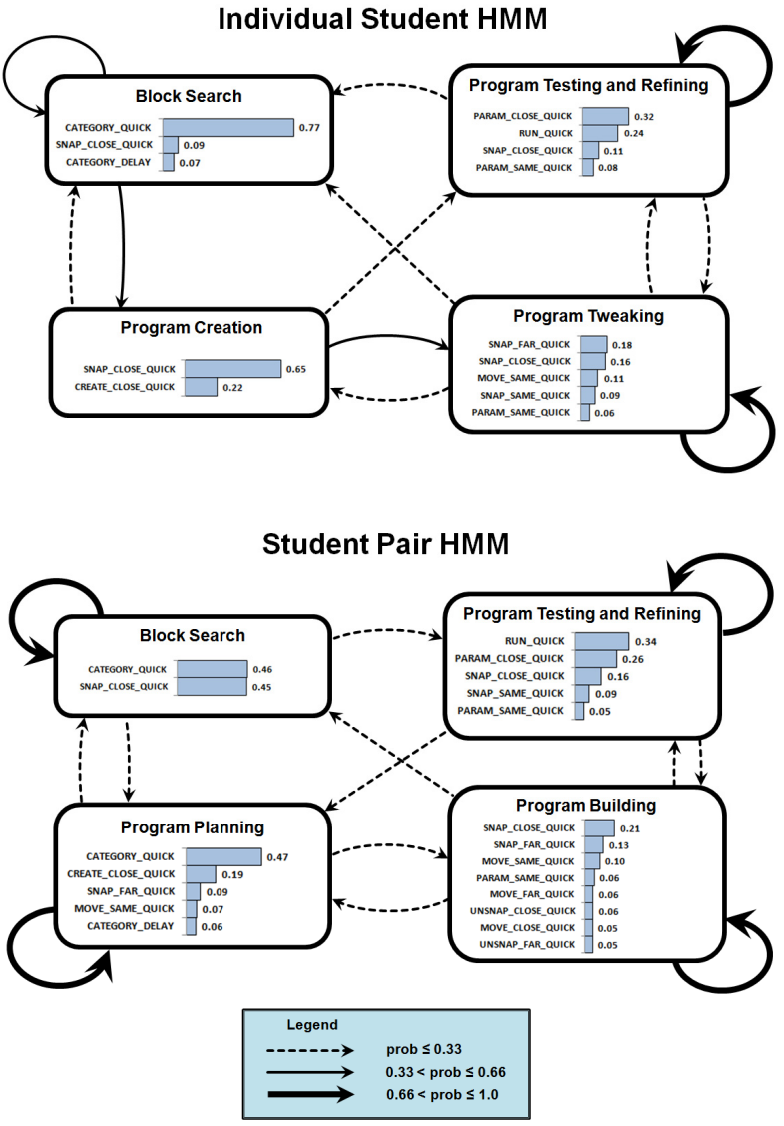

Fig. 3. Student HMMs

Table 2. State frequency summary

Ind. state	μ	σ	Min	Max	Pair state	μ	σ	Min	Max
Search	0.27	0.07	0.17	0.44	Search	0.27	0.08	0.12	0.37
Test/Refine	0.20	0.05	0.12	0.28	Test/Refine	0.24	0.07	0.08	0.33
Tweaking	0.37	0.11	0.11	0.53	Planning	0.20	0.06	0.15	0.35
Creation	0.16	0.03	0.12	0.21	Building	0.29	0.08	0.12	0.37

State Composition. Both models exhibited *block search* and *program testing and refining* states. *Search* was characterized by high probability of category switching within 30 seconds of prior events. For individual students, however, there was also a small probability of category switching after longer than 30 seconds from the previous event, perhaps in part because individual students did not have a partner assisting in their search. Additionally, although both conditions featured snapping events that brought the program closer to the final solution, the probability was higher for student pairs. The *test/refine* state featured high probability of program run events, as well as parameter editing and block snapping. This could indicate that students tested their programs, found errors, and fixed parameters or block order. One event was present in the pair condition that was not present in the individual: parameter editing with no change in distance to final solution. This may be due to partners having conflicting ideas for correcting errors, trying multiple values before finding the correct one.

Individuals exhibited two additional states: *program tweaking* and *program creation*. *Tweaking* consisted of quick block snapping with varying effects on distance to the final solution, as well as block movement and parameter editing with no effect on distance to the final solution. This interpretation was chosen due to the opposing nature of the snapping events: snapping blocks and moving both closer and farther from the final solution. This seems contradictory at first, but may indicate that students were putting together their programs and modifying the block layout to make corrections. *Creation* included block creation and snapping, both bringing the program closer to the final solution. Students spent the least amount of time on average in this state, and the self-transition probability was nearly zero. This suggests that the state serves more as a transition, connecting block search, program tweaking, and testing and refinement.

The pairs also had two additional states: *program planning* and *program building*. *Planning* featured category switching, similar to the search state, but also included events for block creation, movement, and snapping, all with varying effects on distance to the final solution. When a block is created but not snapped, it is placed in the scripting area; similarly, movement involves grabbing and dropping a block within the area. This suggests that the students may be rearranging the blocks' layout to help visualize a proposed solution. *Building* included a large number of prominent observations: snapping, unsnapping, and moving blocks, and editing parameters, all with varying effects on distance to the final solution. This appears to serve a similar purpose to the tweaking state for individuals.

An additional difference between the individual and pair HMMs involves the state transition probabilities: pairs had higher self-transition probabilities than the individuals

for common states (*Search*: pair=0.76, ind.=0.49; *Test*: pair=0.76, ind.=0.67) and in general. This could indicate that pairs persisted with a single problem-solving mode longer than the individuals. Interesting to consider is the difference between the individuals' *tweaking* state and the pairs' *planning* state. Both appear to be states in which students explore solution alternatives, but individuals appear to be more willing to snap blocks when modifying their program, while pairs preferred to move blocks within the scripting area; this could be a key difference between the conditions.

7 Conclusion

During problem solving, adaptive systems can provide a customized learning experience that is likely very different for individuals and pairs. This paper has presented work toward understanding the different problem-solving modes that individuals and pairs utilize. The results show that student pairs appear to persist in a mode more often than individual students. Depending on a task's learning goals, it may be useful to help students persist in a particular problem-solving mode, which could be achieved through the use of scaffolding, for example.

Moving forward, an ITS could use information about an individual's or a pair's current problem-solving mode to provide adaptive support. For example, if the system hypothesizes that students are spending too long in a state of *program tweaking*, it could prompt them to move into a *planning* state with tailored feedback. Additionally, by updating the HMMs with new interaction sequences that include intelligent support, a system could continuously adapt to new modes that may emerge. These types of models can serve as the basis for the next generation of ITSs that support collaborative problem solving, and the methodology could potentially be generalized to support other problem-solving domains.

Acknowledgements. Thanks to the members of the LearnDialogue Group and the Center for Educational Informatics at NCSU for their helpful input. Thanks to Alexandria Vail, Xiaolong Li, and Allison Martínez-Arocho for their contributions. This material is based upon work supported by the National Science Foundation through a graduate research fellowship and IIS-1409639. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Guin, N., Lefevre, M.: From a customizable ITS to an adaptive ITS. In: Lane, H., Yacef, K., Mostow, J., Pavlik, P. (eds.) AIED 2013. LNCS, vol. 7926, pp. 141–150. Springer, Heidelberg (2013)
2. Stamper, J.C., Eagle, M., Barnes, T., Croy, M.: Experimental Evaluation of Automatic Hint Generation for a Logic Tutor. IJAIED **22**(1), 3–17 (2013)

3. Lazar, T., Bratko, I.: Data-driven program synthesis for hint generation in programming tutors. In: Trausan-Matu, S., Boyer, K.E., Crosby, M., Panourgia, K. (eds.) ITS 2014. LNCS, vol. 8474, pp. 306–311. Springer, Heidelberg (2014)
4. Kazi, H., Haddawy, P., Suebnukarn, S.: Leveraging a domain ontology to increase the quality of feedback in an intelligent tutoring system. In: Aleven, V., Kay, J., Mostow, J. (eds.) ITS 2010, Part I. LNCS, vol. 6094, pp. 75–84. Springer, Heidelberg (2010)
5. Walker, E., Walker, S., Rummel, N., Koedinger, K.R.: Using problem-solving context to assess help quality in computer-mediated peer tutoring. In: Aleven, V., Kay, J., Mostow, J. (eds.) ITS 2010, Part I. LNCS, vol. 6094, pp. 145–155. Springer, Heidelberg (2010)
6. Ogan, A., Walker, E., Aleven, V., Jones, C.: Toward supporting collaborative discussion in an Ill-Defined domain. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) ITS 2008. LNCS, vol. 5091, pp. 825–827. Springer, Heidelberg (2008)
7. Ogan, A., Walker, E., Baker, R.S.J.d., Rebolledo-Mendez, G., Jimenez Castro, M., Laurentino, T., de Carvalho, A.: Collaboration in cognitive tutor use in latin america: field study and design recommendations. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1381–1390 (2012)
8. Olsen, J.K., Belenky, D.M., Aleven, V., Rummel, N.: Using an intelligent tutoring system to support collaborative as well as individual learning. In: Trausan-Matu, S., Boyer, K.E., Crosby, M., Panourgia, K. (eds.) ITS 2014. LNCS, vol. 8474, pp. 134–143. Springer, Heidelberg (2014)
9. Williams, L., Wiebe, E., Yang, K., Ferzli, M., Miller, C.: In Support of Pair Programming in the Introductory Computer Science Course. Comput. Sci. Educ. **12**(10), 197–212 (2002)
10. Koedinger, K.R., Stamper, J.C., McLaughlin, E.A., Nixon, T.: Using data-driven discovery of better student models to improve student learning. In: Lane, H., Yacef, K., Mostow, J., Pavlik, P. (eds.) AIED 2013. LNCS, vol. 7926, pp. 421–430. Springer, Heidelberg (2013)
11. Kardan, S., Roll, I., Conati, C.: The usefulness of log based clustering in a complex simulation environment. In: Trausan-Matu, S., Boyer, K.E., Crosby, M., Panourgia, K. (eds.) ITS 2014. LNCS, vol. 8474, pp. 168–177. Springer, Heidelberg (2014)
12. Tenison, C., MacLellan, C.J.: Modeling strategy use in an intelligent tutoring system: implications for strategic flexibility. In: Trausan-Matu, S., Boyer, K.E., Crosby, M., Panourgia, K. (eds.) ITS 2014. LNCS, vol. 8474, pp. 466–475. Springer, Heidelberg (2014)
13. Jeong, H., Gupta, A., Roscoe, R., Wagster, J., Biswas, G., Schwartz, D.L.: Using hidden markov models to characterize student behaviors in learning-by-teaching environments. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) ITS 2008. LNCS, vol. 5091, pp. 614–625. Springer, Heidelberg (2008)
14. McDowell, C., Werner, L., Bullock, H., Fernald, J.: The effects of pair-programming on performance in an introductory programming course. In: Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002), pp. 38–42 (2002)
15. Braught, G., MacCormick, J., Wahls, T.: The benefits of pairing by ability. In: Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE 2010), pp. 249–253 (2010)
16. Thomas, L., Ratcliffe, M., Robertson, A.: Code warriors and code-a-phobes: a study in attitude and pair programming. In: Proceedings of the 34rd ACM Technical Symposium on Computer Science Education (SIGCSE 2003), pp. 363–367 (2003)
17. Martinez, R., Wallace, J.R., Kay, J., Yacef, K.: Modelling and identifying collaborative situations in a collocated multi-display groupware setting. In: Biswas, G., Bull, S., Kay, J., Mitrovic, A. (eds.) AIED 2011. LNCS, vol. 6738, pp. 196–204. Springer, Heidelberg (2011)

18. Martinez-Maldonado, R., Kay, J., Yacef, K.: An automatic approach for mining patterns of collaboration around an interactive tabletop. In: Lane, H., Yacef, K., Mostow, J., Pavlik, P. (eds.) AIED 2013. LNCS, vol. 7926, pp. 101–110. Springer, Heidelberg (2013)
19. Harvey, B., Mönig, J.: Bringing "No Ceiling" to scratch: can one language serve kids and computer scientists? In: Constructionism, pp. 1–10 (2010)
20. Rabiner, L.R., Juang, B.H.: An Introduction to Hidden Markov Models. IEEE ASSP Mag. **3**, 4–16 (1986)
21. Sabourin, J., Rowe, J.P., Mott, B.W., Lester, J.C.: When off-task is on-task: the affective role of off-task behavior in narrative-centered learning environments. In: Biswas, G., Bull, S., Kay, J., Mitrovic, A. (eds.) AIED 2011. LNCS, vol. 6738, pp. 534–536. Springer, Heidelberg (2011)